พอยน์เตอร์ในภาษาปาสคาล

ตอน พอยน์เตอร์กับลิงค์ลิสต์

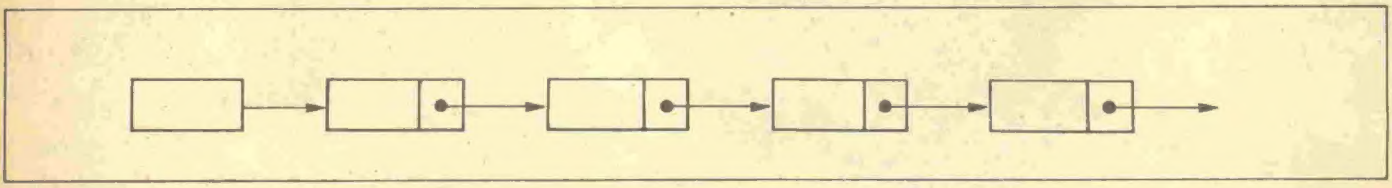
บุญเลิศ เอี่ยมทัศนา

การใช้งานพอยน์เตอร์ในภาษาปาสคาลอย่างแท้จริงคือกับข้อมูลชนิดไดนามิกส์ โดยเฉพาะกับข้อมูลแบบไดนามิกส์ที่เป็นโครงสร้าง คำว่าโครงสร้างหมายถึงว่าข้อมูลแต่ละตัวจะมีความสัมพันธ์กัน ซึ่งมีอยู่หลายรูปแบบ รูปแบบที่ง่ายที่สุดคือลิงค์ลิสต์ ผมจึงเลือกมาเป็นตัวอย่างเพราะเมื่อเข้าใจรูปแบบหนึ่งแบบใดแล้วก็จะทำความเข้าใจกับรูปแบบอื่นได้ไม่ยาก

ลิงค์ลิสต์ (Link List)

ลักษณะของข้อมูลแบบนี้ดังแสดงในรูปที่ 1 คือมีลักษณะเหมือนกระบวนรถไฟ โบกี้หนึ่ง ๆ คือข้อมูลตัวหนึ่ง ๆ โดยที่ข้อมูลแต่ละตัวจะเป็นแบบเรคอร์ด แยกออกเป็น 2 ส่วนคือ ส่วนข้อมูลซึ่งจะเป็นข้อมูลแบบใดก็ได้ตั้งแต่อักขระจนถึงอะเรย์และเรคอร์ด กับส่วนที่เป็นพอยน์เตอร์

ข้อมูลแบบนี้จะเริ่มด้วยพอยน์เตอร์ตัวหนึ่งดังในรูปได้กำหนดเรียกเป็น head โดยที่ head จะชี้ไปยังข้อมูลตัวแรก ในข้อมูลตัวแรกจะมีพอยน์เตอร์ชี้ไปยังข้อมูลตัวที่สอง ในข้อมูลตัวที่สองจะมีพอยน์เตอร์ชี้ไปยังข้อมูลตัวที่สามและต่อ ๆ ไป (เทียบได้กับกระบวนรถไฟ) ดังนั้นหากต้องการดำเนินการกับข้อมูลตัวที่เท่าไรก็ตาม จะต้องเริ่มสาวจาก head ไปเสมอ คือจะต้องดำเนินการในแบบซีเควนเชียล (sequential) เท่านั้น



รูปที่ 1 ลักษณะของข้อมูลแบบลิงค์ลิสต์แต่ละบล็อกคือข้อมูลหน่วยหนึ่ง ๆ

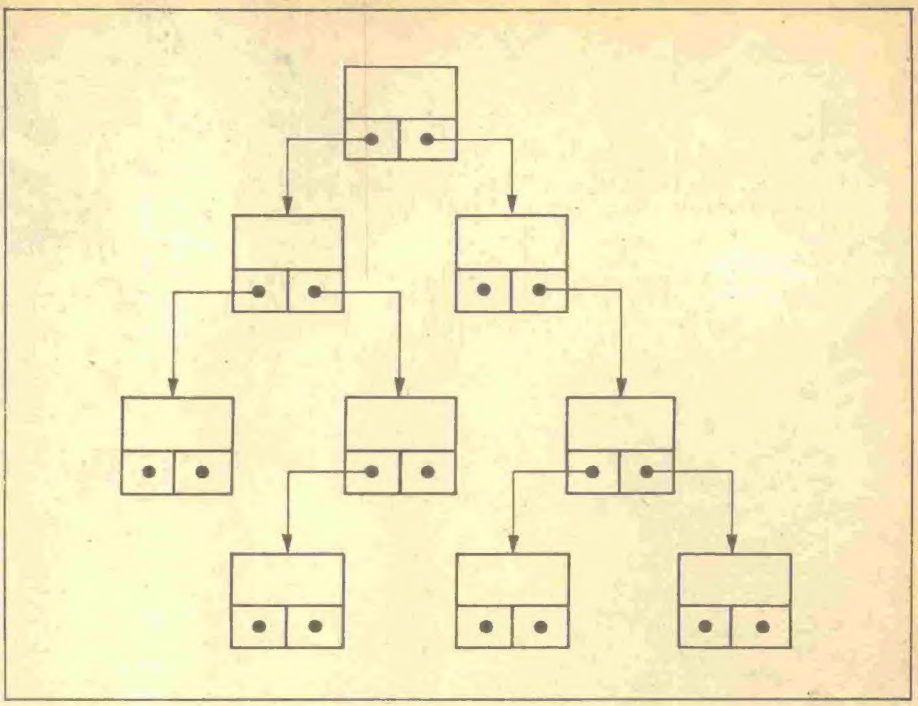
เท่าที่กล่าวมานี้อาจเห็นว่าสู้ข้อมูลแบบอะเรย์ไม่ได้ เพราะอะเรย์ดำเนินการได้ทั้งในแบบซีเควนเชียล และแรนดอม (Random) ทำให้เข้าถึงข้อมูลได้เร็วกว่า ดำเนินกรรมวิธีกับข้อมูลได้เร็วกว่า เรื่องนี้ไม่เป็นจริงเสมอไปครับ กล่าวคือ

ถ้าเป็นการดำเนินการที่ใช้อินเด็กซ์ได้จะดำเนินการกับข้อมูลแบบอะเรย์ได้เร็วกว่า ถ้าเป็นการค้นหาเช่นค้นหาชื่อบุคคลข้อมูลแบบอะเรย์อาจเร็วกว่า แต่ถ้าเป็นการเรียงลำดับข้อมูล (sorting) หรือการแทรก หรือตัดข้อมูล การดำเนินการกับข้อมูลแบบลิงค์ลิสต์จะทำได้เร็วกว่าโดยเฉพาะกับข้อมูลที่แต่ละหน่วยมีขนาดใหญ่ เพราะในลิงค์ลิสต์เราไม่ต้องเคลื่อนย้ายข้อมูลไปมา เพียงแต่เปลี่ยนค่าในพอยน์เตอร์เท่านั้น

นอกจากนี้ข้อมูลชนิดนี้ ยังแตกเป็นรูปแบบอื่นอีกมาก เช่น แบบทรี (tree) ดังแสดงในรูปที่ 2 ซึ่งจะค้นหาข้อมูลได้เร็วกว่าแบบอะเรย์มาก

อีกเรื่องหนึ่งที่ผมยังไม่ได้กล่าวถึงคือ เมื่อเราใช้คำสั่ง New นั้น การกำหนดเนื้อที่ในหน่วยความจำไม่ได้กำหนดในส่วนที่เก็บข้อมูล ดังเช่นอะเรย์ซึ่งจะเป็นการจองเนื้อที่ในหน่วยความจำส่วนนี้ไว้เป็นประจำ หากกำหนดในหน่วยความจำที่ว่างอยู่เรียกว่าฮีพ (Heap) ซึ่งจะกำหนดเมื่อต้องการใช้และยกเลิกได้เมื่อไม่ต้องการใช้

ดังนั้น ในโปรแกรมที่ดำเนินการกับข้อมูลหลายกลุ่ม ไม่พร้อมกันสามารถจะผลัดกันใช้ฮีพได้ ซึ่งถ้าใช้อะเรย์หน่วยความจำอาจไม่พอใจ โดยเฉพาะเมื่อข้อมูลต่างแบบกัน อีกประเด็นหนึ่งคือ ในการใช้ข้อมูลแบบอะเรย์นั้นเราจะต้องกำหนดขนาดคือจำนวนช่องของอะเรย์ไว้แน่นอน ซึ่งจะทำให้มีช่องว่างเมื่อจำนวนข้อมูลน้อยกว่าที่กำหนด และเกิดปัญหาเมื่อจำนวนข้อมูลมากกว่าที่กำหนด ส่วนลิงค์ลิสต์ไม่ขึ้นกับจำนวนข้อมูลจะมีเท่าไรก็ได้จนกว่าจะเต็มหน่วยความจำ



รูปที่ 2 ลักษณะของข้อมูลแบบไบนารี่ทรี (binary tree)

ลิงค์ลิสต์ต้นแบบ

การทำงานของลิงค์ลิสต์อย่างง่าย ๆ คือ เรื่องของแฟ้มบุคคล เช่นเก็บข้อมูลเกี่ยวกับชื่อ ที่อยู่ และหมายเลขโทรศัพท์ของแต่ละบุคคลไว้ในไฟล์ ไม่กำหนดจำนวนบุคคล การดำเนินการคืออ่านข้อมูลนี้ลงหน่วยความจำโดยให้เก็บอยู่ในลิงค์ลิสต์ เพื่อดำเนินการอื่นต่อไป เช่น ค้นหาชื่อ ที่อยู่ หมายเลขโทรศัพท์ เพิ่ม เปลี่ยนแปลงข้อมูล หรือตัดออก เมื่อจบงานแล้วให้เก็บลงไฟล์ตามเดิม

แต่เพื่อให้ตัวอย่างกะทัดรัดและง่ายต่อความเข้าใจ จะสมมติว่าข้อมูลมีแต่ชื่อเพียงอย่างเดียว ซึ่งจะให้ป้อนจากคีย์บอร์ด และสำหรับตัวอย่างแรกนี้จะไม่ดำเนินการใด เมื่อผู้ใช้ป้อนเสร็จแล้วจะให้นำออกแสดงเท่านั้น

แบบของข้อมูล

ได้แสดงการกำหนดแบบของข้อมูลเพื่อจะใช้กับลิงค์ลิสต์ดังแสดงในรูป A1 จะเห็นว่าได้กำหนดแบบของพอยน์เตอร์ pntrT ให้เป็นพอยน์เตอร์สำหรับข้อมูลแบบ personT เสร็จแล้วกำหนดแบบข้อมูล personT เป็นแบบเรคอร์ด (ซึ่งจะต้องเป็นแบบเรคอร์ดเสมอ) ประกอบด้วยฟิลด์ข้อมูล name เป็นสตริงขนาด 30 อักขระ (ฟิลด์ข้อมูลจะมีกี่ฟิลด์ก็ได้) กับฟิลด์พอยน์เตอร์ next เป็นแบบ pntrT เพื่อไว้ชี้ข้อมูลตัวต่อไป (พอยน์เตอร์อาจมีหลายตัวได้เช่นกัน)

```
PROGRAM  LinkList(input,output);
CONST    NONE = '';
TYPE     nameT = String[30];
         pntrT = ^personT;
         PersonT = RECORD
            name : nameT;
            next : pntrT;
                   END;
VAR      head : pntrT;
         name : nameT;
         done : Boolean;
```

A1

```
PROCEDURE  HeadInsert(VAR h : pntrT; nme : nameT);
VAR     n : pntrT;
BEGIN
  New(n);
  n^.name := nme;
  n^.next := h;
  h := n;
END;
```

A2

```
BEGIN  { MAIN1 }
  ClrScr;
  head := nil;
  REPEAT
    Write('Enter name: ');  ReadLn(name);
    IF  name <> NONE  THEN  HeadInsert(head,name);
  UNTIL  name = NONE;
  DisplayList(head);
END.
```

A4

ในการกำหนดตัวแปรได้กำหนดตัวแปร head ไว้เป็นพอยน์เตอร์ที่ตั้งชื่อนี้เพื่อเตือนใจเราว่าจุดประสงค์จะใช้ตัวแปรนี้ชี้ที่ต้นลิสต์เสมอดังแสดงในรูปที่ 1 และคำว่าต้นลิสต์ไม่จำเป็นต้องเป็นข้อมูลแรกที่ป้อน (หรืออ่านจากไฟล์) สุดแล้วแต่การจัดการกับลิสต์ ซึ่งในโปรแกรมนี้ข้อมูลตัวสุดท้ายจะเป็นต้นลิสต์ (คือเรียงกลับลำดับกับที่ป้อน)

การสร้างลิสต์

โปรแกรมที่ใช้ในการสร้างลิสต์คือ HeadInsert อยู่ในรูป A2 และขอให้ดูโปรแกรมที่ใช้งานมันคือโปรแกรมหลักในรูป A4 ด้วยครับ ในโปรแกรมหลักจะให้ head เป็น nil คือไม่ให้ชี้ไปที่ใดเลย สำหรับใช้เป็นเงื่อนไขต่อไปด้วย

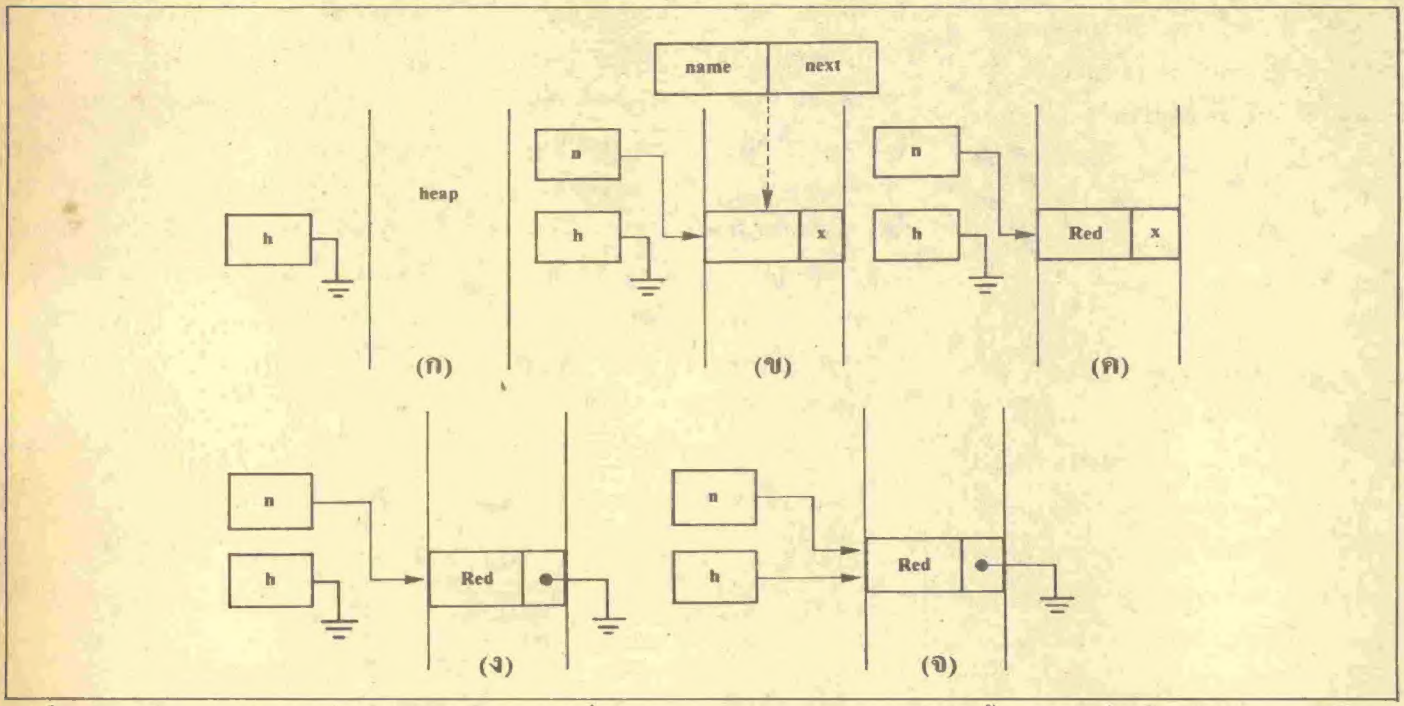
เมื่อโปรแกรมหลักเรียกใช้ Head Insert ในครั้งแรกนั้น Head มีค่าเป็น nil และในฮีพยังว่างเปล่า ส่วน name มีข้อมูลตามแต่ผู้ใช้จะป้อนสมมติว่าเป็น Red การทำงานของโปรแกรม Head Insert ดังแสดงในรูปที่ 3 ซึ่งมีรายละเอียดดังนี้

(ก) เมื่อเริ่มเรียกโปรแกรมนี้ ที่ทำเครื่องหมายไว้เป็น h ลงดิน แสดงว่าไม่ได้ชี้ไปที่ใด ส่วนใน heap ว่างเปล่า

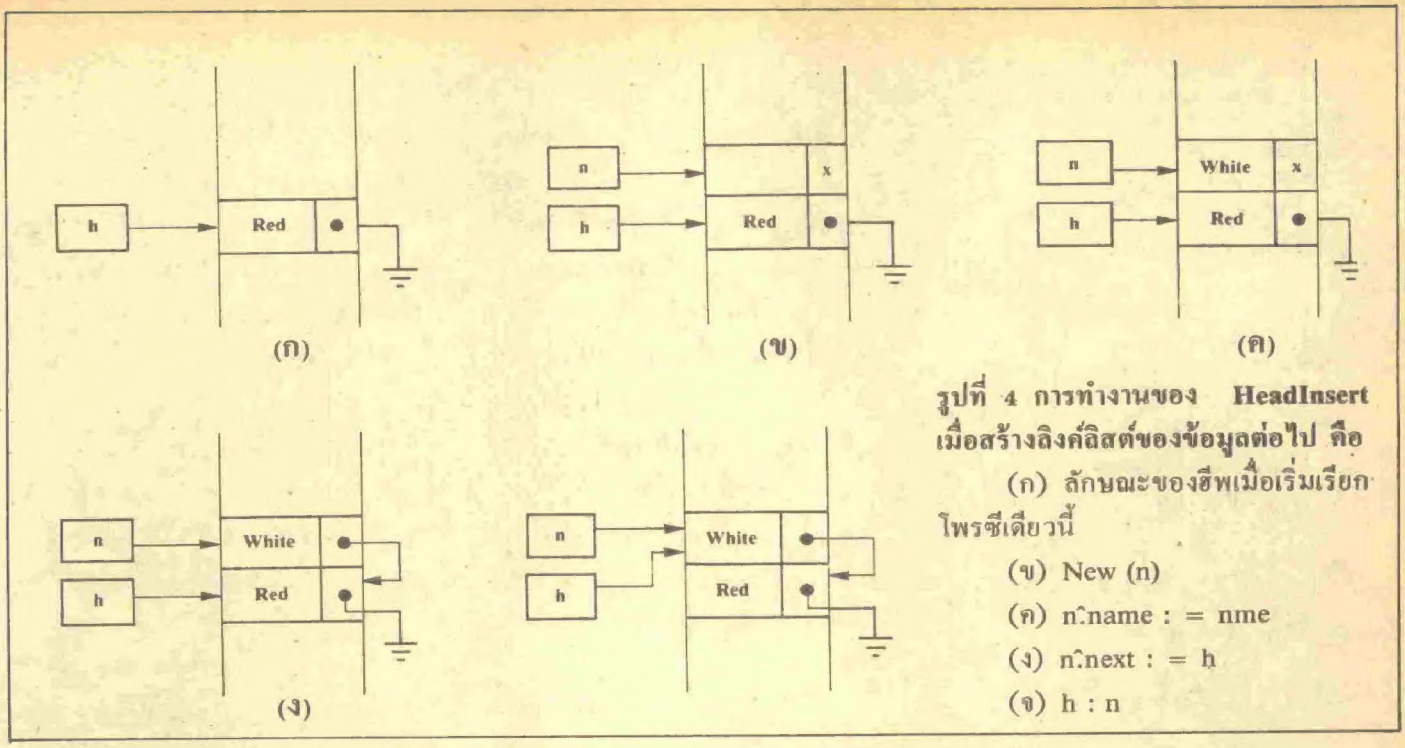
(ข) คำสั่ง New (n) จะเป็นการกำหนดเนื้อที่ใน heap ตามแบบข้อมูล personT ซึ่งแบ่งเป็นส่วนข้อมูลและพอยน์เตอร์ ค่าในพอยน์เตอร์ที่ให้ไว้เป็น x แสดงว่าไม่ทราบว่าชี้ไปที่ใด (เป็นค่าที่ตกค้างอยู่ในหน่วยความจำ) พร้อมกันนี้ก็จะให้ n ชี้มาที่หน่วยความจำนี้

(ค) คำสั่ง n^.name := nme เป็นการนำค่าในตัวแปร nme ให้แก่ฟิลด์ name ที่พอยน์เตอร์ n ชี้อยู่ จะเห็นว่าใช้รูปแบบของคำสั่งเช่นเดียวกับเรคอร์ด

(ง) คำสั่ง n^.next := h ขอ



รูปที่ 3 แสดงการสร้างลิงค์ลิสต์ตามลำดับของคำสั่งในโปรแกรม Head- Insert ดังนี้ (ก) เมื่อเริ่มเรียกโปรแกรมนี้ เป็นครั้งแรก (ข) New (n) ลูกศรประแสดงว่าข้อมูลนี้แบ่งเป็นฟิลด์ name ซึ่งเป็นข้อมูลกับฟิลด์ next ซึ่งเป็นพอยน์เตอร์ (ค) n^.name : = nme (ง) n^.next : = h (จ) h : = n



รูปที่ 4 การทำงานของ HeadInsert เมื่อสร้างลิงค์ลิสต์ของข้อมูลต่อไป คือ

(ก) ลักษณะของฮีพเมื่อเริ่มเรียกโพรซีเดียวนี้

(ข) New (n)

(ค) n^.name : = nme

(ง) n^.next : = h

(จ) h : n

ทบทวนว่า next เป็นตัวแปรพอยน์เตอร์ ค่าที่จะให้แก่ตัวแปร next ได้ต้องเป็นพอยน์เตอร์ด้วยกัน ซึ่งที่จริงก็คือค่าแอดเดรส ในครั้งแรกนี้ h มีค่าเป็น nil (คือเป็นศูนย์) จึงทำให้ next ไม่ชี้ไปที่ใด

(จ) คำสั่ง h : = n หมายถึงให้ค่าพอยน์เตอร์ (ค่าแอดเดรส) ใน n แก่ h ทำให้ h ชี้ไปที่ข้อมูลนี้ ส่วน n ยังคงชี้ไปที่ข้อมูลนี้เช่นกัน (แต่จะถูกยกเลิกโดยปริยายเมื่อกลับไปโปรแกรมหลัก) ส่วน h เนื่องจากเป็นการผ่านค่าในแบบ pass by reference จึงทำให้ head ในโปรแกรมหลักมีค่าเดียวกันคือชี้ที่หน่วยความจำนี้

รูปที่ 4 แสดงการดำเนินการเมื่อผู้ใช้ป้อนข้อมูลที่สอง สมมติว่าเป็น White การดำเนินการจะคล้ายกับที่กล่าวมา ที่แตกต่างกันคือ เมื่อถึงคำสั่ง

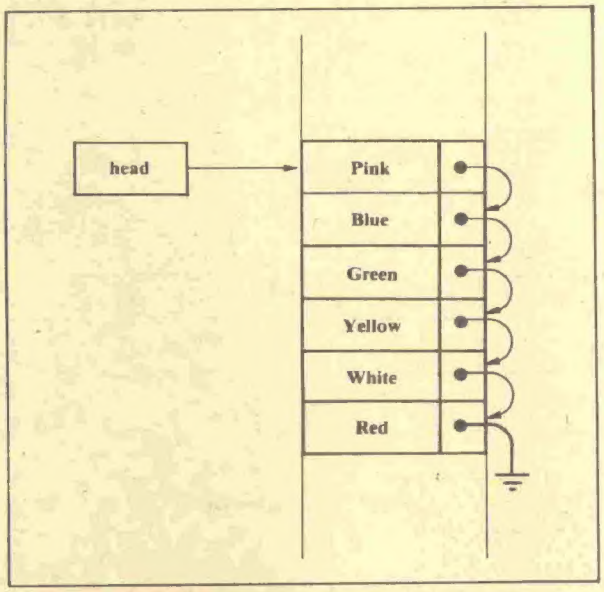
n^.next : = h เนื่องจาก h ชี้ไปที่ข้อมูลแรกคือ Red (จากการผ่านค่า) จึงทำให้ข้อมูลใหม่นี้ชี้ไปยังข้อมูลแรก

h : = n เหมือนเดิมคือให้ h ชี้ที่ข้อมูลใหม่ เกิดเป็นรูปแบบของลิงค์ลิสต์

เมื่อผู้ใช้ป้อนข้อมูลมากขึ้นก็จะเกิดเป็นลิงค์ลิสต์ที่ยาวขึ้นดังแสดงในรูปที่ 5 เทียบได้กับในรูปที่ 1 จะเห็นว่าการต่อข้อมูลจะเป็นการแทรกตรง head คือเมื่อต่อแล้วจะเป็นต้นของลิสต์

นำออกแสดง

เพื่อให้เห็นว่าการทำงานของโปรแกรม HeadInsert ได้นำข้อมูลไปต่อกันเป็นลิงค์ลิสต์จริง ๆ จึงได้ให้นำออกแสดงด้วยโปรแกรม DisplayList รูป A3 ในโปรแกรมเริ่มด้วยวงรอบ WHILE h< >nil คือหากยังไม่สุดลิงค์ลิสต์นั่นเอง ให้นำข้อมูลออกแสดง คำสั่ง h : = h^.next จะทำให้ h ชี้ไปที่ข้อมูลถัดไปในลิสต์ ผลการรันโปรแกรมนี้ทั้งหมด



รูปที่ 5 แสดงภาพเปรียบเทียบของลิงค์ลิสต์ในฮีพ

```
PROCEDURE  DisplayList(h : pntrT);
BEGIN
  WriteLn('LISTING --->');
  WHILE  h <> nil  DO
    BEGIN
    WriteLn(h^.name);
    h := h^.next;
    END;
END;
```

A3

```
Enter name: Red
Enter name: White
Enter name: Yellow
Enter name: Green
Enter name: Blue
Enter name: Pink
Enter name:
LISTING --->
Pink
Blue
Green
Yellow
White
Red
```

รูปที่ 6 ผลการรันโปรแกรมลิงค์ลิสต์

ดังแสดงในรูปที่ 6 จะเห็นว่าข้อมูลเรียงลำดับสลับที่กันกับเมื่อป้อน

ลิสต์แบบเรียงลำดับข้อมูล

ผมจะปรับปรุงโปรแกรมแรกนี้ให้เป็นการเรียงลำดับข้อมูลตามลำดับอักษรไปในตัว เช่น เมื่อป้อน Red เป็นข้อมูลตัวแรก หากต่อไปป้อน White จะนำไปต่อไว้หลัง Red หากป้อน Green จะนำไปไว้หน้า Red ถ้าอยู่ระหว่างกลางจะนำไปแทรก ดังนั้น เมื่อป้อนข้อมูลแต่ละตัวจะพิจารณาได้เป็น 4 ลักษณะ คือ

1. เป็นข้อมูลตัวแรก นำไปเป็นข้อมูลตัวแรก
2. เป็นข้อมูลที่อยู่ลำดับก่อนข้อมูลตัวแรก ให้แทรกหน้าข้อมูลตัวแรก
3. เป็นข้อมูลที่อยู่ลำดับหลังข้อมูลตัวแรก ซึ่งจะต้องค้นเทียบกับข้อมูลตัวถัดไปจน
   - พบข้อมูลที่อยู่ลำดับหลังข้อมูลใหม่ ให้แทรกหน้าข้อมูลนี้
   - สุดลิงค์ลิสต์ ให้นำไปต่อท้าย
4. เป็นข้อมูลที่ซ้ำกับข้อมูลเดิม ซึ่งจะไม่เพิ่มในลิงค์ลิสต์ และจะแจ้งให้ผู้ใช้ทราบ

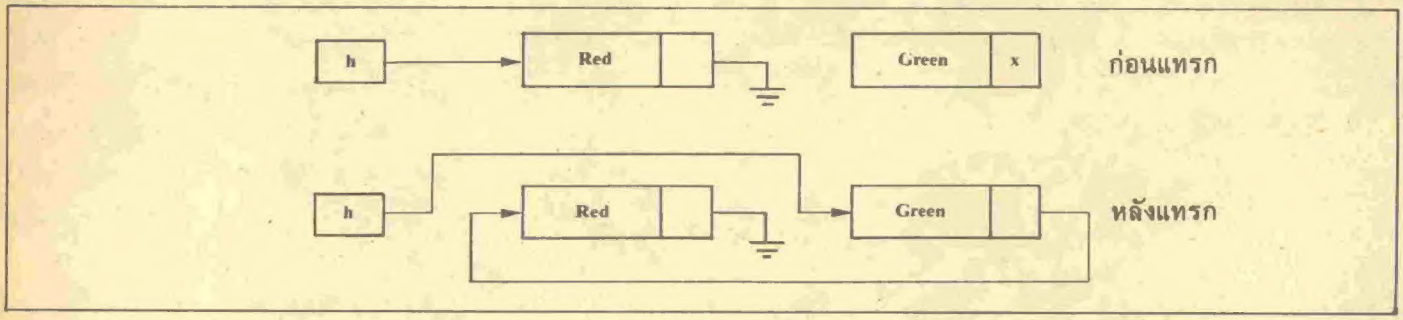
จากลักษณะทั้ง 4 แบบนี้ หากเราเลื่อนพอยน์เตอร์ตัวหนึ่งแทน head ไปจนกว่าจะเข้าลักษณะดังกล่าว แล้วดำเนินการแทรกโดยใช้ HeadInsert ตรงนี้ จะรวมเป็นลักษณะงานได้ 2 แบบคือ

1. แทรกเมื่อ head เป็น nil คือเมื่อเป็นข้อมูลตัวแรก หรือสุดลิสต์
2. แทรกเมื่อ head ไม่เป็น nil คือเมื่อแทรกหน้าข้อมูลตัวแรก หรือแทรกหน้าข้อมูลตัวใดตัวหนึ่งในลิสต์

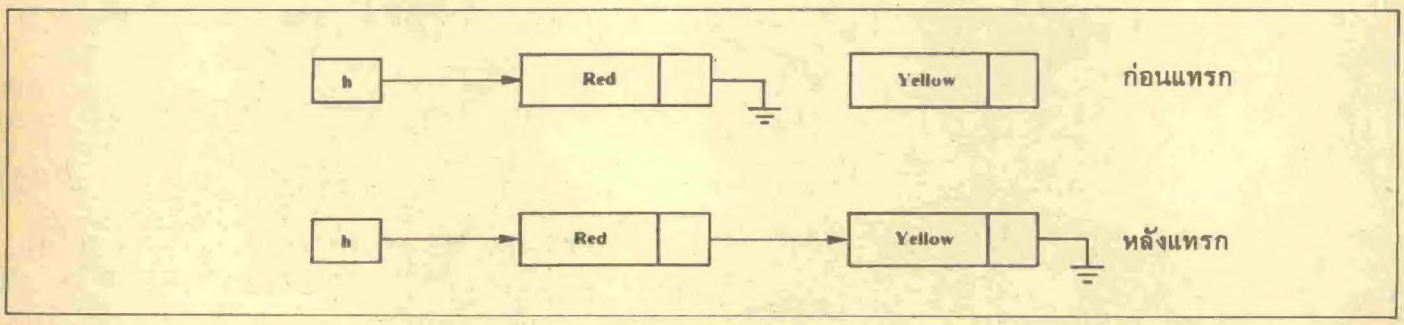
การแทรกข้อมูลในลิสต์ได้แสดงวิธีการไว้ในรูปที่ 7,8 และ 9 คำว่าหน้าหรือหลังนับจาก head คือ h นำไปเขียนเป็นโพรซีเดอร์ ToInsert รูป A5 โดยให้โปรแกรมหลักผ่านค่า head มาให้แก่ h และข้อมูลแก่ nme ส่วน OK มีไว้เพื่อส่งค่ากลับไปว่าได้มีการดำเนินการกับลิสต์หรือไม่

โปรแกรมนี้เขียนในแบบเรียกตนเอง (recursion) การทำงานจึงแบ่งเป็นสองตอน (ตามลักษณะของโปรแกรมเรียกตนเองคือ เมื่อเรียกกับเมื่อส่งกลับ) ก็ขอให้ทำความเข้าใจเป็น 2 ขั้นตอนด้วยครับ ในตอนแรกนี้ขอให้ทำความเข้าใจว่าจะนำข้อมูลไปแทรกข้างหน้าได้อย่างไร โดยยังไม่ต้องพะวงว่าจะต่อเชื่อมกับข้อมูลก่อนหน้ามันเช่นจะต่อ Red เข้ากับ White ได้อย่างไร การทำงานของโปรแกรมเริ่มจากให้ OK เป็นจริงและตรวจสอบเงื่อนไขว่า

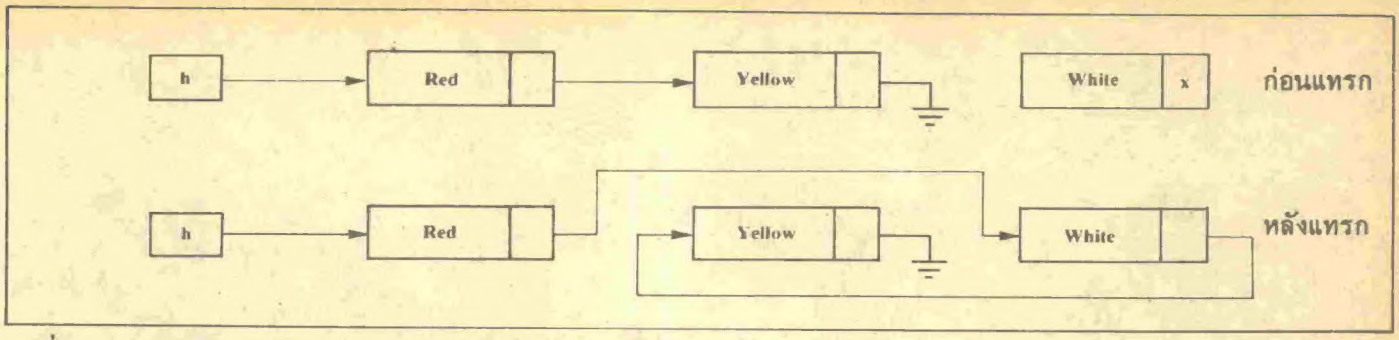
- เงื่อนไขแรก ถ้า h เป็น nil คือเมื่อเป็นข้อมูลแรก ให้นำไปไว้ในลิสต์ดังเช่นในโปรแกรมที่แล้ว
- เงื่อนไขที่สองตรวจว่า ข้อมูล



รูปที่ 7 แสดงการแทรกข้อมูล Green ไว้หน้า Red



รูปที่ 8 แสดงการแทรกข้อมูล Yellow ไว้หลัง Red



รูปที่ 9 แสดงการแทรกข้อมูล White ไว้หน้า Yellow (หลัง Red)

```
PROCEDURE  ToInsert(VAR h : pntrT; nme : nameT; VAR OK : Boolean);
BEGIN
  OK := TRUE;
  IF  h = nil  THEN  HeadInsert(h, nme)
  ELSE  IF  nme = h^.name  THEN  OK := FALSE
  ELSE  IF  nme < h^.name  THEN  HeadInsert(h, nme)
  ELSE  ToInsert(h^.next,nme,OK);
END;
```

A5

นี้ซ้ำกับข้อมูลต้นลิสต์หรือไม่ ถ้าซ้ำให้ OK เป็นเท็จ และไม่ดำเนินการใดกับลิสต์

● เงื่อนไขที่สามให้ตรวจว่า ข้อมูลใหม่มีลำดับที่ก่อนข้อมูลต้นลิสต์หรือไม่ เช่น ตามรูปที่ 9 หากใช่ให้นำไปแทรกไว้หน้าข้อมูลต้นลิสต์โดยใช้ HeadInsert เช่นกัน

● หากไม่เป็นตามที่กล่าวมา ข้อมูลใหม่จะมีลำดับที่หลังข้อมูลต้นลิสต์ ให้เรียกตนเองโดยผ่านค่า h^.next ไปให้แก่ h ทำให้ h ชี้ที่ข้อมูลถัดไปและถือว่าข้อมูลนี้เป็นต้นลิสต์ ดังแสดงในรูปที่ 10

การดำเนินการเช่นนี้จะเหมือนกับตัดข้อมูลต้นลิสต์เดิมไป(ซึ่งได้กล่าวแล้วว่าอย่าเพิ่งสนใจ)สำหรับการทำงานเมื่อเรียกตนเอง จะเป็นไปตามที่กล่าวมาแต่ลักษณะเปลี่ยนไปคือ

● เงื่อนไขแรก จะเป็นไปตามรูปที่ 10 บน ซึ่งในโปรแกรมจะถือว่าเป็นข้อมูลแรก แต่ของจริงเป็นการต่อท้ายลิสต์

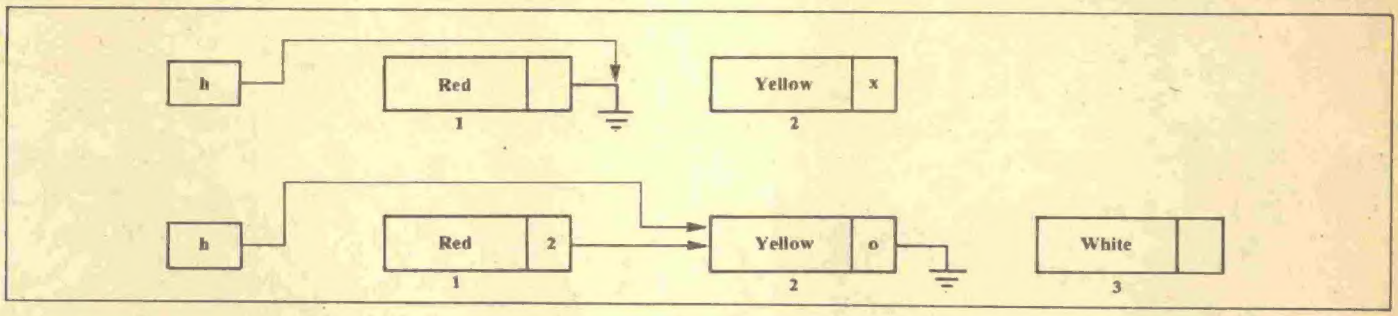
● เงื่อนไขที่สอง หากเรียกตนเองต่อ ๆ ไปจะกลายเป็นการเทียบกับข้อมูลทุกตัวในลิสต์

● เงื่อนไขที่สาม จะเป็นไปตามรูปที่ 10 ล่าง ซึ่งในโปรแกรมถือว่าเป็นการแทรกหน้า Yellow (หรือหน้าข้อมูลหากมีการเรียกตนเองต่อไป)

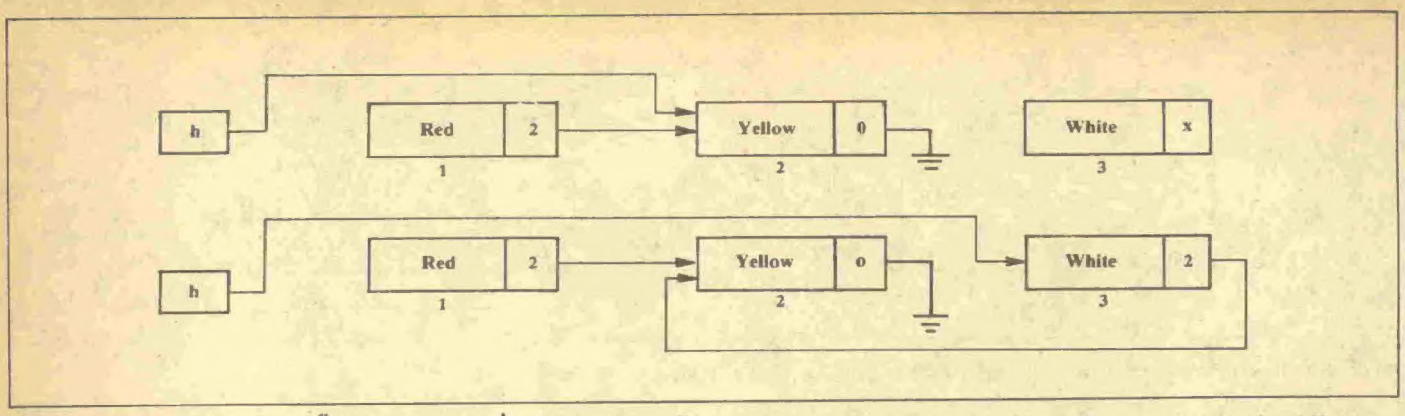
การดำเนินการเมื่อเรียกใช้ Head Insert แล้วจะเป็นดังรูปที่ 11 **ต่อไปจะเป็นการส่งกลับ** ขอให้ดูในรูปที่ 10 ล่าง แต่สมมติผู้ใช้ป้อนข้อมูลที่ 3 เป็น yellow แทนที่จะเป็น White สำหรับตัวเลขสมมติว่าเป็นแอดเดรส

เมื่อผู้ใช้ป้อน Yellow (ซ้ำ) การทำงานของ ToInsert ตรวจว่า Yellow มีอันดับที่หลัง Red จะเรียกตนเองโดยผ่านค่า next ของ Red คือ 2 ให้แก่ h ทำให้ h มีค่าเป็น 2 ชี้ที่ Yellow(เดิม) เมื่อตรวจเทียบพบว่าซ้ำกันจะไม่ดำเนินการกับลิสต์ ตอนส่งกลับจะส่งค่า 2 ใน h ให้ next ของ Red ส่วน h ให้ค่า 1 ของตนคืนมา ทำให้ลิงค์ลิสต์คงสภาพเดิม

แต่เมื่อผู้ใช้ป้อน White ปรากฏว่ามีลำดับที่ก่อน Yellow จึงแทรกไว้หน้า Yellow ทำให้ h มีค่าเป็น 3 (ขอให้ดูในโปรแกรม HeadInsert) เมื่อส่งกลับจะเป็นการส่งค่า 3 ใน h ให้แก่ next ของ Red ทำให้มันชี้ไปยังข้อมูล White แทนที่จะเป็น Yellow ส่วน h คงได้ค่า 1 ของ



รูปที่ 10 แสดงลักษณะการชี้ข้อมูลของ h เมื่อเรียกตนเองครั้งแรก โดยที่รูปบนเป็นการแทรกของข้อมูลตามรูปที่ 8 ส่วนรูปล่างเป็นการแทรกของข้อมูลตามรูปที่ 9 ตัวเลขที่เขียนไว้จะกล่าวถึงภายหลัง



รูปที่ 11 แสดงลักษณะการชี้ข้อมูลของ h เมื่อเรียกตนเองครั้งแรกรูปบน ตรวจพบว่า White มาก่อน Yellow จึงเรียกใช้ HeadInsert ได้ผลดังรูปล่าง (ก่อนการส่งกลับ)

ตนคืนมาเช่นเดิม ทำให้ลิงค์ลิสต์เปลี่ยนเป็นดังรูปที่ 9

นั่นคือเราสามารถจะป้อนข้อมูลได้เรื่อย ๆ โดยที่โปรแกรมนี้จะนำไปเก็บในลิสต์แบบเรียงลำดับข้อมูลให้

ตามที่กล่าวมานี้บางท่านอาจสงสัยว่าทำไมผมถึงต้องตรวจสอบว่า h เป็น nil หรือไม่ก่อน หรืออาจสงสัยว่าทำไมผมไม่เขียนเงื่อนไขเป็น

IF (h = nil) OR (nme<h^.name) THEN

ทั้งนี้เพราะว่าเมื่อใดก็ตามที่เราใช้ h^.name คือใช้ h^ หากh เป็น nil จะเกิด run time error และในโปรแกรมตัวแปลภาษาปาสคาลส่วนมาก เมื่อเราให้ตรวจสอบเงื่อนไขหลายเงื่อนไขด้วยการใช้ AND หรือ OR ดังตัวอย่าง มันจะตรวจสอบเงื่อนไขทุกเงื่อนไขไม่ว่าจะได้ผลเมื่อตรวจเงื่อนไขแรกแล้วหรือไม่ จึงต้องแยกกัน (สำหรับเทอร์โบปาสคาล 4.0 ทำได้โดยใช้ compiler directior B-)

ถอนข้อมูลจากลิสต์

การดำเนินการกับลิสต์มีเรื่องหลัก ๆ อีกหนึ่งเรื่องคือ การค้นหาข้อมูล (search) ซึ่งอาจจะดำเนินการเพื่อนำข้อมูลออกแสดงทางจอภาพรวมทั้งเครื่องพิมพ์ หรือเพื่อแก้ข้อมูลในบางฟิลด์ เช่นแก้ที่อยู่ หมายเลขโทรศัพท์ หรือเพื่อถอนข้อมูลนั้นออกจากลิสต์ทั้งเรคอร์ด

ทั้งหมดนี้มีกรรมวิธีคล้าย ๆ กัน แต่การถอนข้อมูลจากลิสต์มีเรื่องที่จะต้องดำเนินการกับลิสต์มากกว่า จึงขอนำเรื่องนี้มากล่าวถึง เช่น ตามตัวอย่างที่ยกมานี้จะให้ผู้ใช้ป้อนชื่อบุคคลที่ต้องการถอนออกจากลิสต์ นำมาพิจารณาดำเนินการได้เป็น 3 ลักษณะคือ

1. ผู้ใช้ถอนข้อมูลจากลิสต์ว่าง ซึ่งถอนไม่ได้จึงไม่ถอน และแจ้งให้ผู้ใช้ทราบ

2. ไม่มีข้อมูลเดิมในลิสต์ เช่นกันจะถอนไม่ได้จึงไม่ถอน และแจ้งให้ผู้ใช้ทราบ

3. มีข้อมูลในลิสต์ ซึ่งจะต้องดำเนินการถอนออกจากลิสต์

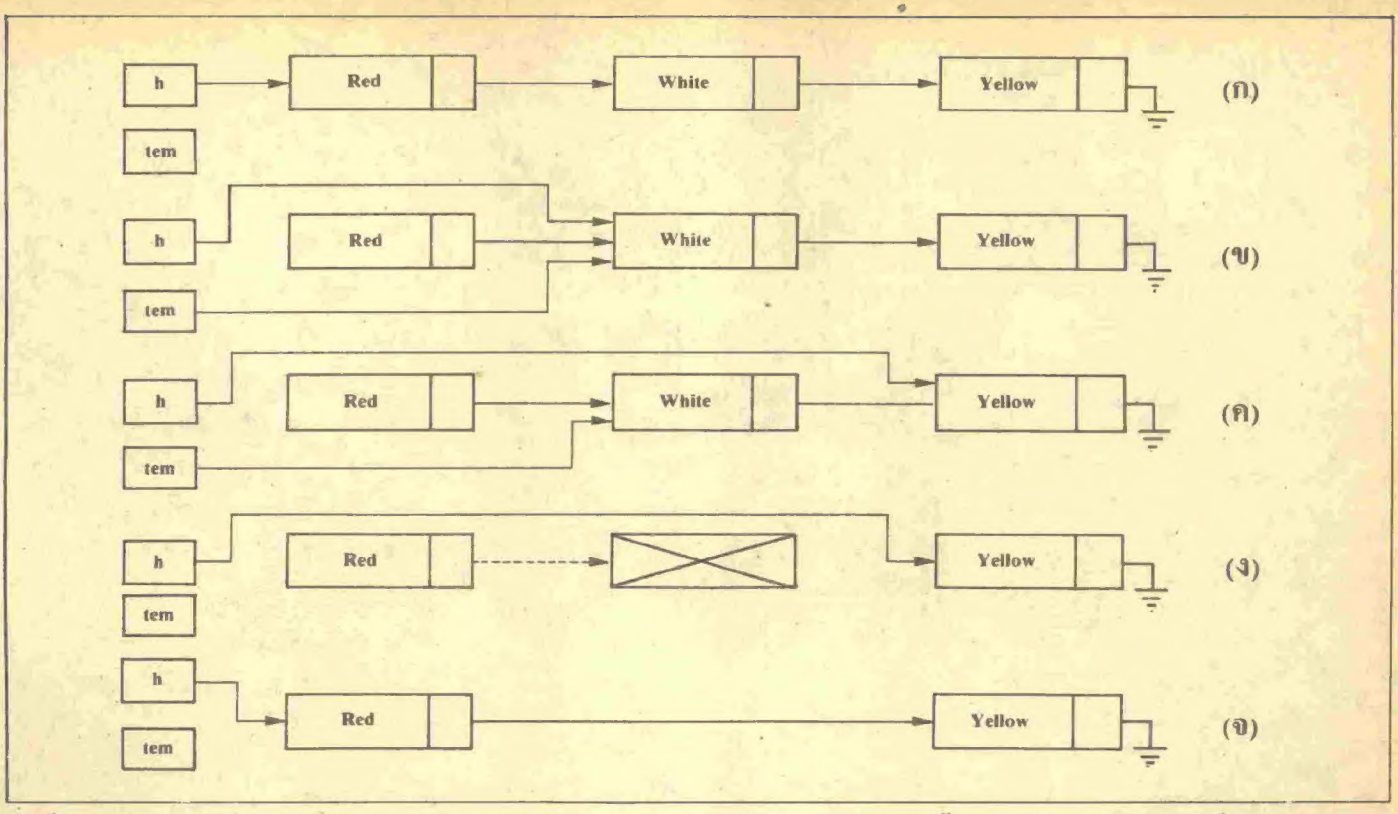
การถอนข้อมูลในลิสต์จะทำได้โดยตรวจหาข้อมูลที่ต้องการถอนในลิสต์จนพบ แล้วให้ข้อมูลตัวก่อนมันชี้ไปยังข้อมูลตัวหลังมัน คือชี้ข้ามตัวที่จะถูกถอน เสร็จแล้วให้ยกเลิกข้อมูลตัวที่ต้องการถอน

แต่ผมจะเขียนโปรแกรมในแบบเรียกตนเองคือโปรแกรม ToDelete ตามรูป A6 ซึ่งจะขออธิบายควบคู่ไปกับรูปที่ 12 กล่าวคือ สมมติว่ามีข้อมูลอยู่ 3 ตัว คือ Red, White กับ Yellow และผู้ใช้ต้องการถอนข้อมูล White การดำเนินจะเป็นดังนี้

ในการตรวจข้อมูลเที่ยวแรกไม่พบข้อมูล White จึงเรียกตนเอง ทำให้ h ชี้ไปที่ White เมื่อตรวจเทียบปรากฏว่าตรงกัน ให้ OK เป็นจริงและให้ tem ชี้ไปที่ White ด้วย จากนี้ก็ให้ h ชี้ข้อมูลถัดไปคือ Yellow เสร็จแล้วยกเลิกข้อมูล White

```
PROCEDURE  ToDelete(VAR h : pntrT; nme : nameT; VAR OK : Boolean);
VAR  tem : pntrT;
BEGIN
  IF  h = nil  THEN  OK := FALSE
  ELSE  IF nme = h^.name  THEN
    BEGIN  OK := TRUE;  tem := h;
           h := h^.next;  Dispose(tem)  END
  ELSE  ToDelete(h^.next, nme, OK);
END;
```

A6



รูปที่ 12 แสดงการถอนข้อมูล White ออกจากลิสต์ตามโปรแกรม TryDelete ดังนี้ (ก) ข้อมูลเดิม (ข) เมื่อเรียกตนเอง (ค) เมื่อปฏิบัติคำสั่ง h = h^.next (ง) เมื่อปฏิบัติคำสั่ง Dispose (tem) (จ) เมื่อส่งกลับ

ด้วยคำสั่ง Dispose (tem) เมื่อส่งกลับจะทำให้ Red ชี้ไปที่ Yellow เช่นเดียวกับที่กล่าวในโปรแกรม TryInsert ได้ผลดังแสดงในรูปที่ 12 (จ)

โปรแกรมหลักที่จะทดลองใช้งาน ToInsert และ TryDelete อยู่ในรูป A7 (ขอให้ใช้แทนโปรแกรมหลักในรูป A4) โปรแกรมนี้เพียงแต่ให้ทดลองป้อนชื่อลงลิงค์ลิสต์ และทดลองยกเลิกข้อมูลเท่านั้นครับ ส่วนการดำเนินการจริงจะต้องดำเนินการเกี่ยวกับไฟล์ด้วย แต่ผมคงต้องขอจบเพียงเท่านี้ครับ

```
BEGIN  { MAIN }
  head := nil;
  REPEAT
    Write('Enter name: ');  ReadLn(name);
    IF  name <> NONE  THEN
      BEGIN
      ToInsert(head, name, done);
      IF  NOT done  THEN  WriteLn('  Duplicate name');
      END;
  UNTIL  name = NONE;
  DisplayList(head);

  REPEAT
    Write('Enter name to be delete: ');  ReadLn(name);
    IF  name <> NONE  THEN
      BEGIN
      ToDelete(head, name, done);
      IF  NOT done  THEN  WriteLn('Name not found');
      DisplayList(head);
      END;
  UNTIL  name = NONE;
END.
```

A7